Sequential Equivalence Checking for Clock-Gated Circuits

Hamid Savoj, Alan Mishchenko, and Robert Brayton, *Fellow, IEEE*

***Abstract*—Sequential logic synthesis often leads to substantially easier equivalence checking problems, compared to general-case sequential equivalence checking (SEC). This paper theoretically investigates when SEC can be reduced to a combinational equivalence checking *(*CEC*)* problem. It shows how the theory can be applied when sequential transforms are used, such as sequential clock gating, retiming, and redundancy removal. The legitimacy of such transforms is typically justified intuitively, by the designer or software developer believing that the two circuits reach the same state after a finite number of cycles, and no difference is observed at the outputs due to fanin non-controllability and fanout non-observability effects.**

A method based on these theorems was applied to 12 large industrial examples, which had been combinationally and sequentially clock-gated by a commercial clock-gating tool. These examples lead to SEC problems, which are problematic for a general, state-of-the-art SEC engine. The new approach completed verification of all examples and was about 30 times faster on the 3 cases where the conventional SEC engine was able to prove the problem within one hour.

***Index Terms*—Formal verification, logic synthesis, sequential synthesis, synthesis for low power.**

I. Introduction

A SEQUENTIAL circuit, depicted in Fig. 1, is composed of a combinational logic part (A^1) and a set of memory elements [called flip-flops (FF), or flops].

It also has a set of primary inputs (PIs) and a set of primary outputs (POs). The combinational logic has inputs composed of the PIs plus the current state (CS) of the FF's, and has outputs composed of the POs plus the next state (NS), which is stored in the FF's on the next clock edge.

We assume that the FF's start in a known initial state at time 0. Two sequential circuits are said to be equivalent if, starting from their respective initial states, for any given (infinite) sequences of PIs, the two generated (infinite) sequences of the associated POs are identical.

Combinational synthesis changes only the combinational logic but keeps the PO and NS functions (as functions of PIs and CSs) unchanged, although the network computing these functions can be completely restructured. In this case, equivalence checking consists of checking the equivalence of the combinational parts of the two circuits.

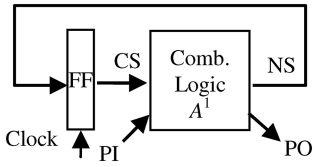

Fig. 1. Sequential circuit A.

In contrast, sequential synthesis additionally uses information about the states unreachable from the initial state to modify the combinational logic further. This can consist of: 1) using the unreached states as don't cares; 2) using the fact that two states are equivalent; 3) using a different state-encoding for the reachable states; 4) retiming the FFs; or 5) merging two signals in the circuit if in all reachable states the signals have the same values.

These two types of synthesis lead to different kinds of problems when comparing the original circuit against the synthesized circuit. While combinational equivalence checking (CEC) is NP-complete, sequential equivalence is PSPACE-complete. This paper considers a particular type of sequential synthesis in which the NS combinational functions can be changed using state equivalence reasoning. We show that the corresponding SEC problem is only NP-complete and give methods for reducing the SEC problem to a CEC problem in these cases. Such synthesis transformations arise frequently in commercial software packages that perform clock gating to reduce power.

To initially motivate this paper, we look at one method for sequential synthesis that is quite practical but not obviously valid. Consider a sequential circuit, A, which is to be optimized by a k-step unrolling process as indicated in Fig. 2. The combinational part of A (called A^1) is copied k times and these are cascaded so that the NS outputs of one copy are feeding into the CS inputs of the following copy, resulting in the combinational circuit C^1 shown in the top part of Fig. 2. Note that the outputs of C^1 are (PO_1, ... , PO_k, and NS).

We use the notation $C^1 = (A^1)^k$ to denote this circuit, which has k copies of A^1. Now suppose combinational synthesis is applied to the logic of the first copy while the other k-1 copies are left untouched, resulting in the circuit $C^2 = B^1(A^1)^{k-1}$ shown at the bottom of Fig. 2. Although the NS outputs of the final copy are necessarily preserved by the combinational

Manuscript received June 30, 2013; accepted September 23, 2013. Date of current version January 16, 2014. This work was supported in part by SRC contract 1875.001 and in part by NSA Grant Enhanced Equivalence Checking in Crypto-Analytic Applications. This paper was recommended by Associate Editor W. Kunz.

The authors are with the Electrical Engineering and Computer Sciences Department, University of California at Berkeley, Berkeley, CA 94720 USA (e-mail: hamid@savojsolutions.com; alanmi@eecs.berkeley.edu; brayton@eecs.berkeley.edu).

Digital Object Identifier 10.1109/TCAD.2013.2284190

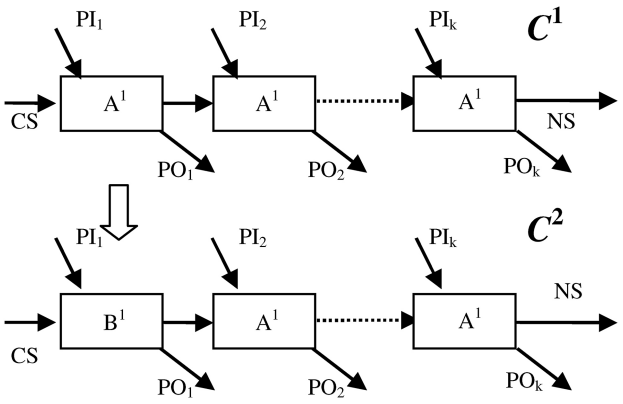

Fig. 2. k-step unrolling, ODC optimization of A^1, and the combinational part of FSM A.

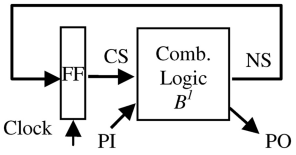

Fig. 3. Derived sequential circuit B.

synthesis, the internal NS output of B^1 as well as the successive NS outputs of the k-2 copies of A^1 in C^2 may be different from the corresponding internal NS outputs in C^1.

The purpose of the last k-1 copies of A^1 in this scenario is to produce observability don't cares (ODCs) [13] used for transforming the first copy of A^1 into B^1, which forms the combinational part of a new sequential circuit B.

Now consider the sequential circuit shown in Fig. 3.

Several questions arise regarding this.

1) Is the derived circuit B sequentially equivalent to A? This question cannot be answered by just comparing A^1 with B^1 because they are not combinationally equivalent. Also, the fact that C^1 is combinationally equivalent to C^2 does not constitute a proof of sequential equivalence.
2) Would comparing $(A^1)^k$ with $(B^1)^k$ suffice for a SEC proof?
3) Suppose A is unrolled n times creating $(A^1)^n$ and the *last* copy of A is synthesized using satisfiability don't cares (SDCs) [13] provided by the first n-1 copies of A^1, creating $(A^1)^{n-1}B^1$. Are A and the resulting B sequentially equivalent? For reasons similar to those listed in Question 1, this is not obvious.
4) More generally, suppose A is unrolled $n+k$ times and the n^{th} copy of A^1 is synthesized, using both ODCs from $(A^1)^k$ and SDCs from $(A^1)^{n-1}$, to produce $(A^1)^{n-1}B^1(A^1)^k$ and the corresponding sequential machine B. Is B sequentially equivalent to A?

Section II answers these questions by proving three theorems. The theorems are stated for general SEC problems and give sufficient CEC test problems, which if true prove the corresponding SEC problems. Generally, theorem 1 might be applicable when a synthesis transform has been used, based on non-observability principles and theorem 2 when non-controllability is the basis of the transform.

Section III demonstrates instances where theorem 1 can be applied while Section IV demonstrates instances where theorem 2 or 3 can be applied. The emphasis is on those scenarios where sequential clock gating methods are used to save power. Such synthesis methods are often used in commercial clock-gating software. In addition, new inductive techniques for removing sequential redundancies are discussed in Sections III and IV. Section V discusses the simultaneous use of both theorems, and gives a counterexample to item four above.

Section VI discusses literature related to this paper, including ATPG methods for sequential redundancy removal. Section VII shows experimental results where the proposed methods are used to formally verify equivalence after sequential clock gating is applied to industrial circuits. The experiments illustrate how the new methods make sequential equivalence checking (SEC) more efficient and practical for such problems. Finally, Section VIII summarizes the paper and poses open questions for future research.

II. Sequential Equivalence

Let A be a sequential circuit and A^1 denote the combinational part of A. Instead of using $(A^1)^k$ to denote A^1 cascaded k times, we just use A^k to denote the combinational circuit obtained by connecting k copies of A^1 at the NS outputs and CS inputs. This is shown on top of Fig. 2. The outputs of A^k are the set of k POs of A, one set for each time-frame plus the final NS signals at the output of the k^{th} frame. The inputs of A^k are the k sets of PIs of each frame, plus the initial CS signals of the first frame.

Let A and B be two sequential circuits with the same PIs and POs, and the same number of FFs. B^nA^k denotes the combinational circuit where the NS outputs of B^n are connected to the CS inputs of A^k. The connection is based on the 1-1 mapping between the FFs of A and B given by the user. The circuit, B^1A^{k-1}, is shown in Fig. 2.

In this paper, it is always assumed that a single initial state is given for a sequential machine. We are not concerned with initializing sequences and so on, but follow the philosophy articulated in [1]. Two machines are sequentially equivalent if starting in their initial states, and their POs produce identical sequences when identical sequences are applied to the PIs.

For two sequential circuits A and B, $A=B$ denotes that the circuits are sequentially equivalent starting from the two given initial states. If C and D are combinational circuits, then $C=D$ denotes that they are combinationally equivalent, i.e., for any input, their outputs are identical.

The first question listed in Section I concerns the equivalence of two related combinational circuits. Specifically, does combinational equivalence of A^{n+k} and B^nA^k imply the equivalence of the sequential machines A and B? The equivalence of combinational circuits B^nA^k and A^{n+k} is depicted in Fig. 4. Notice that if $B^nA^k = A^{n+k}$, then starting from any initial state s_0 and applying any sequence of inputs $I = \{I_1, I_2, \ldots, I_{n+k}\}$, the outputs $O = \{O_1, O_2, \ldots, O_{n+k}\}$ and the final next states s_{n+k} are equal on both sides.

We emphasize that, in order to create the related combinational circuit B^1A^{k-1} from A^1 and B^1, a one-to-one correspondence between the FFs of A and B must exist, so that the appropriate wires can be attached[1]. This also implies that the two circuits have the same state encoding.

Lemma 1: Suppose two sequential circuits A and B have the same PIs and POs. Use some 1-1 mapping between the FF's of A and B to form B^nA^k as shown in Fig. 4. If $B^nA^k = A^{n+k}$, then $B^{n\times p}A^k = A^{n\times p+k}$ for any $p > 0$.

Proof: We use induction on p to prove this. For the base case, clearly this holds for $p = 1$. For the inductive case, assume it is true for p, that is

$$B^{n\times p}A^k = A^{n\times p+k}.$$

Apply n copies of A^1 on the right to each side. Equality still holds since the states and inputs passed to A^n are equal on both sides

$$B^{n\times p}A^kA^n = A^{n\times p+k}A^n.$$

Regroup the A^1's

$$B^{n\times p}A^nA^k = A^{n\times(p+1)+k}.$$

Replace A^nA^k with B^nA^k

$$B^{n\times p}B^nA^k = A^{n\times(p+1)+k}.$$

Regroup the B^1's. Therefore

$$B^{n\times(p+1)}A^k = A^{n\times(p+1)+k}$$

and by induction, $B^{n\times p}A^k = A^{n\times p+k}$ *for any* $p > 0$. ■

Theorem 1: Suppose two sequential circuits A and B have the same PIs and POs. Use some 1-1 mapping between the FFs of A and B to form ${}^nA^k and A^{n+k}. If B^nA^k = A^{n+k}$, then $A = B$, when both are initialized with the same arbitrary initial state.

Proof: Suppose the theorem is false meaning $B^nA^k = A^{n+k}$ but $A \neq B$. Then, there exists a state s_0, an integer m, and a sequence of PIs, $\hat{I} = \{\hat{I}_1, \hat{I}_2, \ldots, \hat{I}_m, \ldots\}$ such that at clock cycle m, one of the POs of A differs from the corresponding PO of B, when both A and B are given the same initial states s_0. We know from Lemma 1 that $B^{n\times p}A^k = A^{n\times p+k}$ when $B^nA^k = A^{n+k}$. This means outputs of A and B are equal for any $p > 1$. Choose p large enough such that $n\times p > m$. This contradicts the existence of sequence $\hat{I} = \{\hat{I}_1, \hat{I}_2, \ldots, \hat{I}_m, \ldots\}$ that causes a difference in outputs. Therefore, $A = B$. ■

Note that nothing is assumed about how B was obtained. It could be the finite-unrolling combinational synthesis process as mentioned in item one of Section I, or a sequential clock-gating method as illustrated in Section III. Also, if $B^nA^k \neq A^{n+k}$ one can try to prove $A = B$ by increasing k or n, and possibly a false negative may go away.

A variation of theorem 1 swaps A and B and states that if $A^nB^k = A^{n+k}$, then $A = B$. However, this conclusion is only true under certain conditions that are considered below.

Let B^r denote the full Boolean space of the r state variables in A. Let R_n^A be states s such that $\exists s_0 \in \mathsf{B}^r$ that can reach s in A after exactly n cycles. Also let $[A = B]_{\hat{S}}$ denote that A is equivalent to B for all initial states $\hat{s} \in \hat{S}$.

[1]In some applications, the number of FFs in the two circuits may not be equal; this is addressed by inserting dummy FFs in one of the circuit.

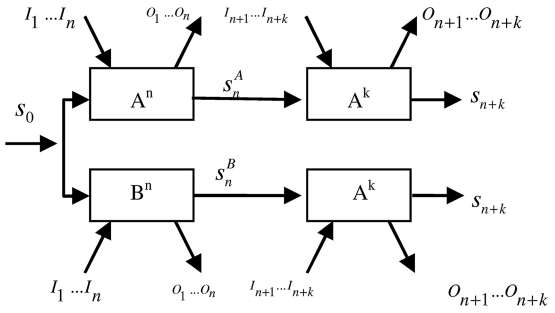

Fig. 4. First n steps in the A and B computations followed by k steps of A.

Lemma 2: $R_{k+1}^A \subseteq R_k^A, \forall k \geq 0$ *where* $R_0^A = \mathsf{B}^r$.

Proof: This is proved by induction on k. Clearly $R_1^A \subseteq \mathsf{B}^r$. Assume $R_k^A \subseteq R_{k-1}^A$. Since R_k^A contains any state that can be reached from some state in R_{k-1}^A in one cycle, and by assumption $R_k^A \subseteq R_{k-1}^A$, then states reachable from R_k^A in one cycle, namely R_{k+1}^A, has to be a subset of R_k^A. Therefore, by induction, $R_{k+1}^A \subseteq R_k^A, \forall k$. ■

Theorem 2: if

$$A^nB^k = A^{n+k} \; then \; [A = B]_{R_n^A}.$$

Proof: Notice that R_n^A are the states that are reached starting from any initial state $\hat{s} \in \mathsf{B}^r$ using an n-step input sequence applied to A^n. The equation $A^nB^k = A^{n+k}$ can also be written as $[B^k = A^k]_{R_n^A}$. We show that if $[B^k = A^k]_{R_n^A}$, then $[B^{pk} = A^{pk}]_{R_n^A}$ for any $p > 0$ using induction. Clearly the base case $p = 1$ holds. Assuming $[B^{pk} = A^{pk}]_{R_n^A}$, add A^k on the right to both sides. Equality still holds since the current states and inputs passed to A^k are equal on both sides of $[B^{pk}A^k = A^{pk}A^k]_{R_n^A}$. Replace $[B^{pk}A^k]_{R_n^A}$ with $[B^{pk}B^k]_{R_n^A}$. This can be done because, given $[B^{pk} = A^{pk}]_{R_n^A}$, starting from a state $s_0 \in R_n^A$, any state s_{pk} reached after $pk > 0$ copies of B^1 is the same state as the one reached after pk copies of A^1. Using Lemma 1, since $s_0 \in R_n^A$ and s_{pk} is a state reachable after A^{pk}, then $s_{pk} \in R_n^A$ and thus $[B^{pk}B^k = A^{pk}A^k]_{R_n^A}$, or $B^{(p+1)k} = A_{R_n^A}^{(p+1)k}$. By induction, $[B^{pk} = A^{pk}]_{R_n^A}, \forall p > 0$, and thus $[A = B]_{R_n^A}$. ■

Fig. 5 illustrates the need to start only on the states reachable by an initial sequence of A^1's. It is easy to check that $A^1A^1 = A^1B^1$ from the STGs shown, but $A \neq B$ because if A and B start in State 01, the PO sequences for A and B are not the same. However, note that if we start from any state that can be reached after one clock cycle of A (i.e., States 00 and 11), then $A = B$.

In cases where the initial set of states is not in R_n^A, a stronger condition is needed for the equality to hold. This happens in the SDC clock gating discussion presented in Section IV. The extra condition is shown in theorem 3.

Let J be a subset of FFs that will be restricted to a given value and let S_J denote the set of states where the J flops are given the specified initial value, while the rest of the flops can have an arbitrary value. The notation $[A = B]_{S_J}$ means machine A and B are equal starting from the given initial set of states S_J.

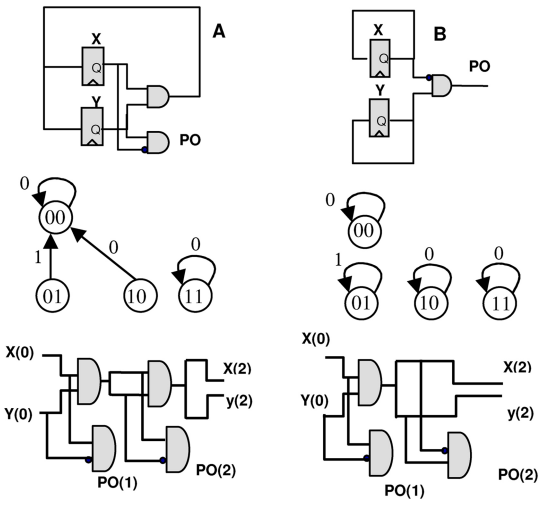

Fig. 5. $A^2 = A^1B^1$, but sequential equivalence occurs only after one cycle of A. A dark bubble at an input to a gate denotes inversion.

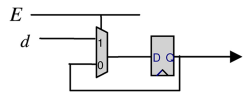

Fig. 6. Equivalent model for clock-gating a flip-flop.

Theorem 3: If $[B^m = A^m]_{S_J}$ for some $m \geq n$ and $[A^n B^k = A^{n+k}]$, then $[A = B]_{R_n^A + S_J}$.

Proof: Since $[B^m = A^m]_{S_J}$, the equality of the POs is assured for the first m steps. Given that $m \geq n$, the state s_m^A reached by both machines at the end of the m^{th} step is in R_n^A, i.e., $s_m^A \in R_n^A$, then the two machines are equal for any subsequent input sequences $\{\hat{I}_{m+1}, \ldots\}$ according to theorem 2. We conclude that $[A = B]_{R_n^A + S_J}$, i.e., equality holds when the machines start from any state in $R_n^A + S_J$. ■

Note that the first condition of theorem 3 states that if we can find an initial state for the J FFs such that $[B^m = A^m]_{S_J}$ holds, then A and B so initialized are sequentially equivalent.

Let R_n^{A,S_J} be the set of states s such that $\exists s_0 \in S_J$ that can reach s in Machine A after exactly n cycles.

Corollary 1: If $[B^1 = A^1]_{S_J + R_1^{A,S_J} + R_2^{A,S_J} + \ldots + R_n^{A,S_J}}$ and $[A^n B^k = A^{n+k}]$, then $[A = B]_{R_n^A + S_J}$.

Proof: Since $[B^1 = A^1]_{S_J + R_1^{A,S_J} + R_2^{A,S_J} + \ldots + R_n^{A,S_J}}$, then $[B^n = A^n]_{S_J}$. Therefore, $[A = B]_{R_n^A + S_J}$ follows from theorem 3. ■

III. Applications of Theorem 1

This section discusses sequential synthesis transforms whose equivalence can be proven using theorem 1.

A. Sequential Clock-Gating

In clock-gating (for power or other purposes), one finds an enable signal, E, and a group of flops such that when the enable is false, the clock to the flops is disabled, ensuring that they keep their old values. This can be modeled with a feedback loop through a multiplexor, as shown in Fig. 6, and thus only regular flops are required in the model.

Fig. 7 shows two sequential circuits, A and B, to be proved equivalent. The designer (or the CAD system) expects that the

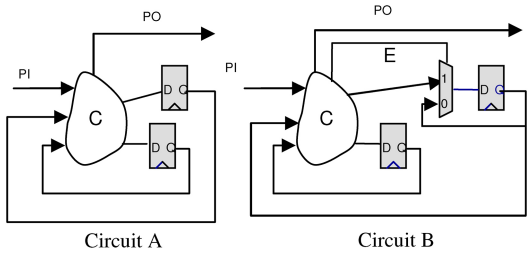

Fig. 7. Two circuits to be verified equivalent when an enabling signal E has been used for clock gating in B.

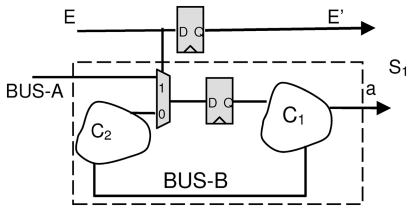

Fig. 8. Removing feedback loops within a sequential circuit S_1. When $E' = 0$, a is not observable.

effect of the enabling signal, E, is never observable in B. It is expected that any perturbation in the circuit caused by the clock gating dies down after k steps.

However, the internal states for the first k-1 steps may be changed. In this case, theorem 1 can be used.[2]

The examples shown in Figs. 8 and 9 illustrate a clock-gating transform whose legitimacy is argued theoretically, but in practice we need to perform equivalence checking. These examples deal with adding and removing redundant loops in a circuit. Fig. 8 shows a sequential circuit S_1that operates as follows. When $E = 1$ (representing start), *BUS-A* provides new values that are loaded into the flops of S_1 as shown. The computation is uninterrupted as long as $E = 0$ and restarts with new loaded values when $E = 1$. Further, it is known that the signal a is observable only when $E' = 1$. E' is just E delayed by 1 cycle (as shown). *BUS-B* is a cut-set representing the internal feedback to the FF's of S_1.

Theorem 4: BUS-B is redundant in Fig. 8.

Proof: (a) When $E(t) = 1$, the state $s(t)$ of S_1 is replaced by the values on *BUS-A*. Thus the internal state, $s(t)$, of S_1 reached at time t is never observed because $E'\ (t+1) = 1$.

(b) When $E(t) = 0$, the result of the computations done in S_1 at t is stored as $s(t+1)$. Note that $a(t+1)$ is not observable because $E(t) = 0$ results in $E'(t+1) = 0$. $a(t+j)$ is observable only if $E(t+j\text{-}1) = 1$ $(= E'(t+j))$. However, if $E(t+j\text{-}1) = 1$, then by the argument in (a), the value of $s(t+j\text{-}1)$ is lost, and hence $a(t+j)$ is independent of $s(t+j\text{-}1)$. Thus any observable output depends on the value loaded in at the previous time frame from *BUS-A* and is not dependent on *BUS_B*. ■

Note that not only can *BUS-B* be removed, but also the logic in C_2 and any FF that do not have a combinational path to *a,*

[2]There are some unpublished methods that obtain the enable signal by induction. These methods depend on the initial state. In these cases, we do not know yet how to do SEC easily, and speculate that probably induction has to be part of the SEC method.

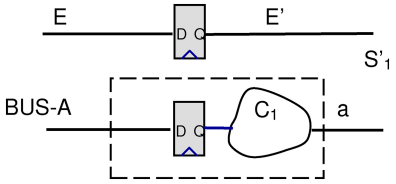

Fig. 9. Sequential circuit S'_1 that can be clock-gated.

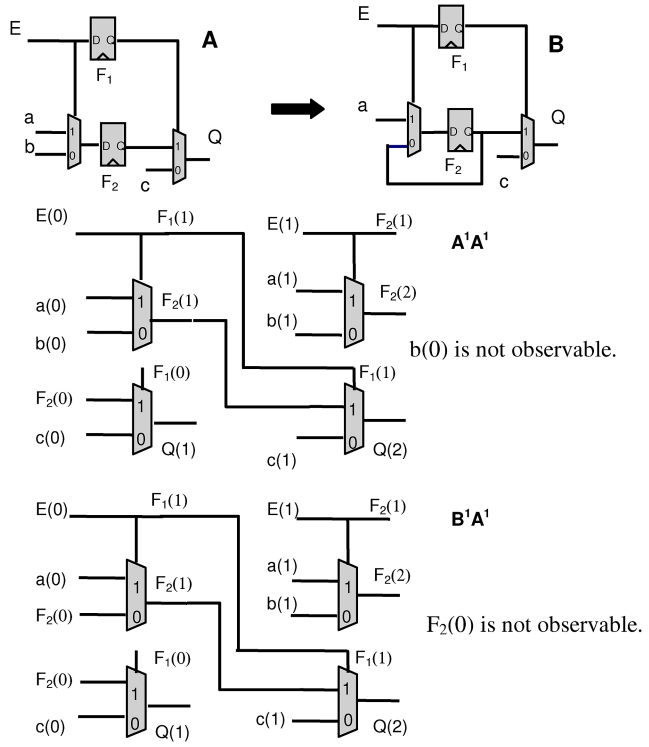

Fig. 10. Loops added to circuit A to create B. Verification using theorem 1 succeeds.

can be removed. Simplification of the circuit in Fig. 8 leads to the one in Fig. 9.

Conversely, we could clock gate the circuit of Fig. 9 to produce a circuit similar to the one in Fig. 8, knowing that a is not observable when $E' = 1$.

B. Asymmetry in Verification

While the circuits in Figs. 8 and 9 are sequentially equivalent, synthesis in one direction represents loop removal and the other direction represents loop addition. It turns out that the addition and removal of loops are not symmetrical operations with respect to checking equivalence using theorem 1.

Consider Fig. 10, where loops are added during synthesis (as shown by the arrow), producing circuit B from A. Applying theorem 1, where $k = 2$, we compare the combinational circuits A^1A^1 with B^1A^1 as shown at the bottom of Fig. 10. The correctness of the addition of the loop can be argued using sequential ODCs [8]. Indeed, theorem 1 works in this case (see comments in Fig. 10).

However, in Fig. 11, a loop is removed from circuit A to create circuit B. Its removal cannot be argued by ODCs, so theorem 1 is not expected to work necessarily, and indeed it does not work here (see comments in Fig. 11). An

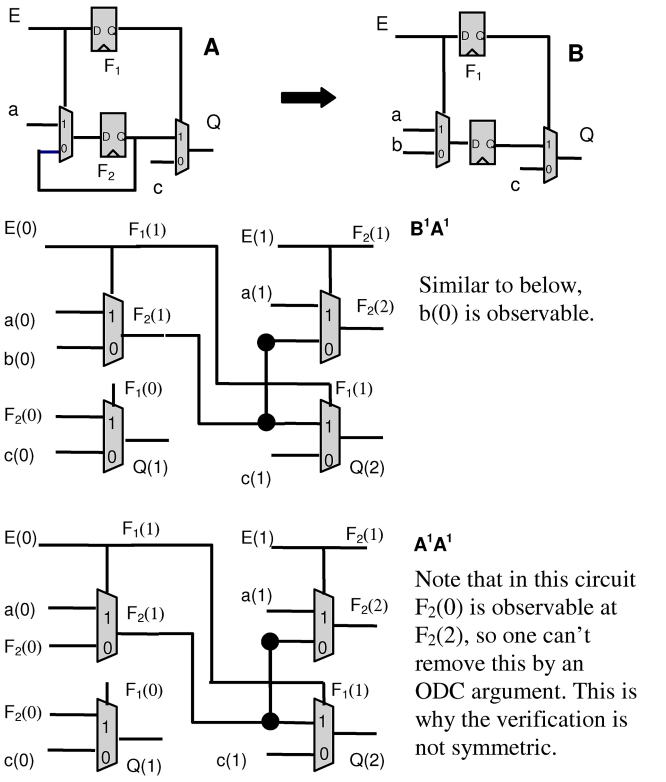

Fig. 11. Loops removed from circuit A to create B. Verification using theorem 1 cannot be used.

inductive approach to remove this kind of loop is discussed in Section III-D.

Of course, equivalence can be proved with theorem 1 by reversing the roles of A and B in Fig. 11, but here we are just illustrating the asymmetry between adding and removing loops with this example.

Theorem 1, which uses $B^n A^k = A^{n+k}$, works when B has additional loops compared to A. However, if there are additional loops in A, they block observability propagation and equivalence does not hold. This asymmetry raises some issues; adding loops is fine for the application of theorems 1 and 2, but removing loops is problematic.

This is not a problem if loops are only added or only removed (one can simply swap the roles of A and B if loops are deleted), but it is a problem if a new circuit, A_S, is obtained from A by removing some loops and adding others. In practice, this can happen when clock gating is inserted in the RTL by the user, but the software has evaluated that this is not economical and has removed it, but has added some different clock-gating. In Section IV, we discuss several ways of handling such situations by using intermediate synthesis results.

C. Adding New Variables to Simplify Verification

Let A be the original circuit and suppose B has been derived from A by adding some loops and removing others. Suppose an intermediate circuit, C, is obtained from A by first removing the undesired loops. Then both A and B can be derived from C by only adding loops. Verification can be done by applying theorem 1 twice to check $A^1C^1 = C^1C^1$ and $B^1C^1 = C^1C^1$.

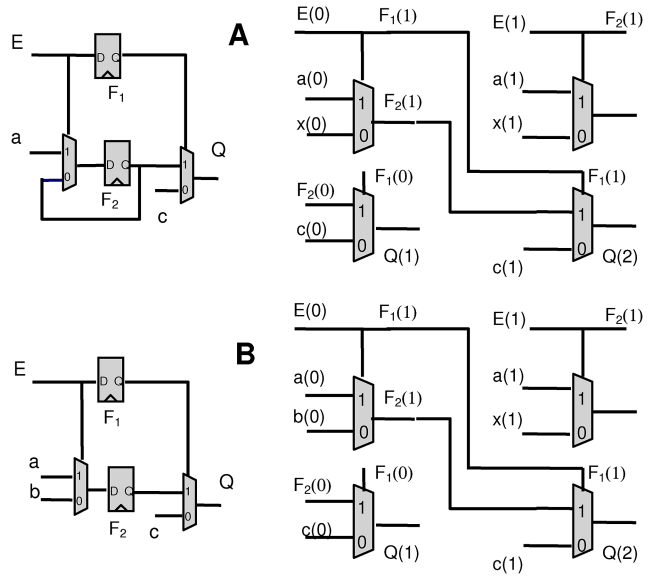

Fig. 12. To prove $A = B$, we check that $\hat{A}^2$ (upper right) equals $B^1\hat{A}^1$ (lower right) by adding free variable x to replace redundant loop in A. Note that $x(0)$ is not observable in the top right.

If both verifications are successful, then $A = B$. However, the disadvantage of this approach might be that two CEC problems have to be solved. Possibly a better approach is to apply the following theorem, which allows for loop removal, but the SEC problem is easier. It is an application of theorem 1 to a modified problem.

Theorem 5: Let circuit A have some feedback loops and let BUS-A be a cut of such loops. Let B be derived from A by removing the loops and adding other loops and let BUS-B be a cut of the added loops in B. Derive $\hat{A}$ from A by replacing BUS-A by free variables x and $\hat{B}$ from B by replacing BUS-B by free variables y. If $\hat{A} = \hat{B}$, then $A = B$.

Proof: Since $\hat{A} = \hat{B}$, the sequential miter, $\hat{A} \oplus \hat{B}$ is UNSAT, for all x, y. Note that B and $\hat{B}$ differ only at the points where the loops were broken, with $\hat{B}$ having the free variable y at the cut. Similarly A and $\hat{A}$ differ but x is the free variable at the cut. The behavior of the sequential miter $\hat{A} \oplus \hat{B}$ is a superset of the behavior of $A \oplus B$ because the cut feedback signals have been replaced by free variables x and y in $\hat{A} \oplus \hat{B}$. Since $\hat{A} = \hat{B}$, then $\hat{A} \oplus \hat{B}$ is UNSAT, and thus $A \oplus B$ must be UNSAT also. ■

This theorem provides a way to remove and add loops and prove equivalence using a single application of theorem 1. Theorem 1 is expected to succeed if the cut signals are redundant while the addition of loops as discussed in Section III is not so problematic and can be justified by sequential observability arguments. Thus, to prove $A = B$ we just have to check that $\hat{B}^1\hat{A}^{k-1} = \hat{A}^k$.

The use of the free variables x and y is illustrated in Fig. 12 where, for simplicity, only loops of A are deleted.

D. Redundancy Removal

Another application of theorem 1 is in removing sequential redundancies. Given an unrolling A^{n+1}, we discuss two kinds of redundancies that can be removed in the first copy of A^1 using ODCs from the last n copies of A^1.

1) The first type is not observable at any of the outputs or the final next state function of A^{n+1}.
2) The second type is observable at the final next state function but not observable at any of the outputs. We will see how this can be used later.

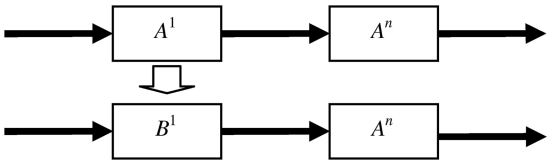

Fig. 13. Redundancy removal using ODCs from A^n to simplify A^1 resulting in B^1, then $A = B$.

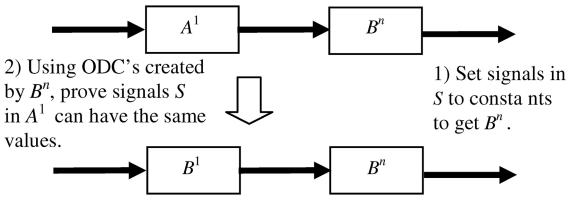

Fig. 14. Redundancy removal for eliminating loops.

Traditional ODC optimization takes care of the first kind of redundancy where ODC's are used to synthesize A^1 from A^n as shown in Fig. 13. Let B^1 be an optimized version of A^1. By construction the changes in B^1 are not observable at the outputs and at the final next-state function of B^1A^n. Therefore, $B^1A^n = A^{n+1}$ and from theorem 1, $A = B$.

However, theorem 1 also supports an approach to remove redundancies of the second type. This is done by an approach where we speculate a signal is constant in the last n copiesA^n. We then prove that it is constant in the first copy.

Theorem 6: Let B^1 be created from A^1 by setting a group of signals S in A^1 to constants. Form A^1B^n (first row of Fig. 14). If, using ODCs from B^n, the signals S in A^1 can be set to the same constant values (last row), then A = B.

Proof: By construction, the change $A^1B^n \rightarrow B^1B^n$ is not observable at any of the outputs or at the final next-state function of B^1B^n. Since optimization converts the first copy of A^1 to B^1 using ODC's from B^n (last row of Fig. 14), then $B^1B^n = A^1B^n$ and from theorem 1, $A = B$. ■

The idea here is that the change to A^1 may be observable at the NS outputs of A^n, but not at the NS outputs of B^n.

An example of this is shown in Fig. 15. It shows how to remove a redundant loop by setting the signal *E-Mux* to 1 in the forward copy of A^1 in A^1A^1 and using ODC's to prove the same signal can be set to 1 in the backward copy. The setting of *E-Mux*(1) = 1 in the forward copy of A^1 makes the condition *E-Mux*(1) = 0 unobservable at $F_2(2)$. As a result, the condition *E-Mux*(0) = 0 is unobservable at all outputs and next states and the signal *E-Mux* can be set to one.

The optimization represented by theorem 6 can be done one signal at a time or using a larger set of signals S. For

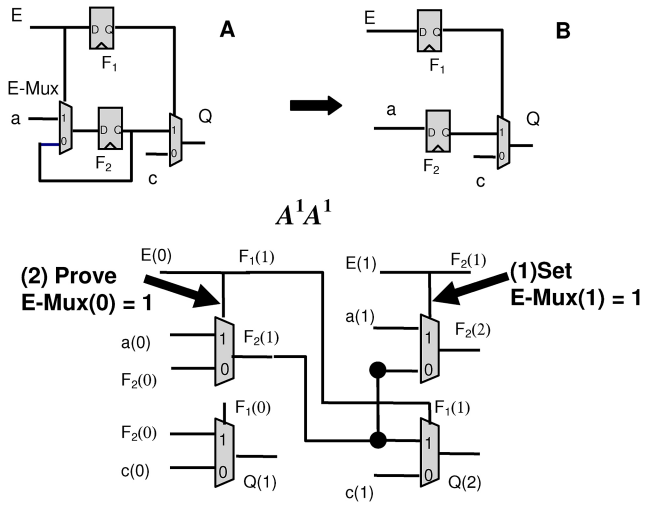

Fig. 15. Removing loops.

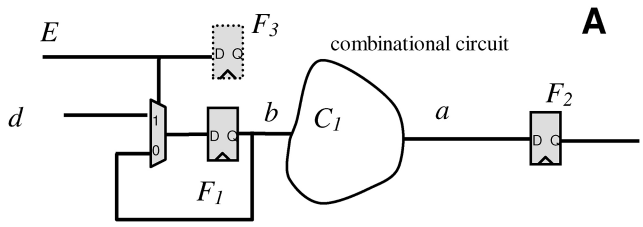

Fig. 16. Dummy flop F3 added to match Fig. 17.

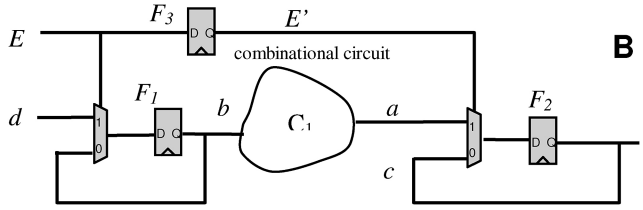

Fig. 17. Delayed enable used to gate F_2.

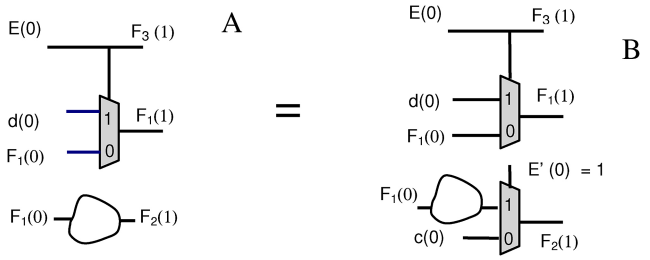

Fig. 18. After one cycle, A and B are equal when F_3 is initialized to 1 (implies E'(0) = 1).

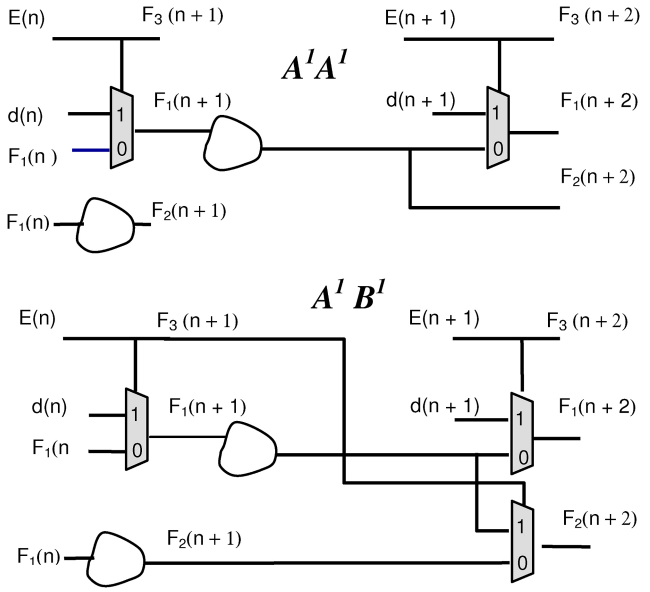

Fig. 19. Checking that $A^1A^1 = A^1B^1$.

example, it can be used to remove all sequential clock gates that are generated using ODCs from a design since all ODC-justified clock gates are redundant sequential loops added to the design, thus, $B^1B^n = A^1B^n$ should hold.

IV. Applications of Theorem 3

This section discusses sequential transformations whose equivalence can be proved by using theorem 3.

A. Sequential Clock-Gating

Figs. 16 and 17 show an example of a circuit (a one-stage pipeline) that has been clock-gated using fanin information. The original circuit is in Fig. 16 and the clock-gated circuit is in Fig. 17. Since the combinational circuit C_1 is only fed by the outputs of F_1, its outputs, *a,* do not change if the outputs of F_1 do not change. This happens when the enable signal E is 0. Therefore, we can clock-gate the subsequent flops, F_2, enabled with a delayed version E.

Because this argument is concerned with the fanin behavior of the newly clocked flop, the equivalence of the original and clock-gated circuits is provable using theorem 3 with $n = m = k = 1$. Note that the original and clock gated designs differ by an additional flop in the clock gated version. We matched this by putting in a dummy flop in the initial circuit fed by signal E but with no fanout. Initializing F_3 to 1 (which is the subset S_I in theorem 3) would take care of the requirement that $\left[B^1 = A^1\right]_{F_3=1}$ because this would cause the first cycle to act as if there were no clock gating. This is shown in Fig. 18.

In Fig. 19, we check that $A^1A^1 = A^1B^1$. Since all the checks hold, theorem 3 implies that $\left[B = A\right]_{F_3=1}$.

B. Pipelines

An important area where sequential clock gating is used is in the context of pipelines. In this section, we illustrate how theorem 3 might be applied to an n-stage pipeline.

Suppose a clock-gating mechanism is provided by an external FSM whose POs are used to control the clock-gating of the FF's between the pipeline stages. The FSM generates proper enabling signals for each stage so that stage k is clock gated one cycle after stage $k-1$. This is illustrated with a three-stage pipeline in Fig. 20 where A is the original pipeline and B the clock-gated version. A simple implementation of such an FSM starts by creating an enable signal (E_1) for the first stage, such that $E_1 = 0$ when inputs at the first stage have not changed after one cycle (i.e., each input matches its value

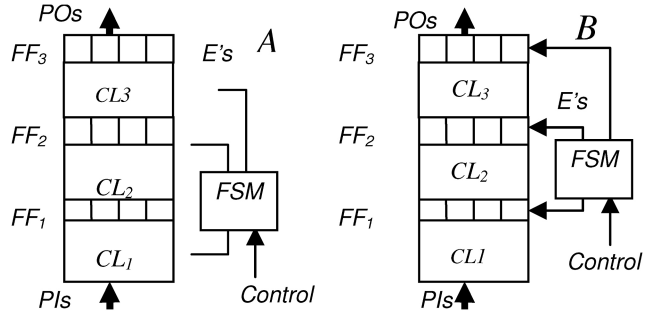

Fig. 20. A is a three-stage pipeline circuit before clock-gating, and B is the clock-gated version.

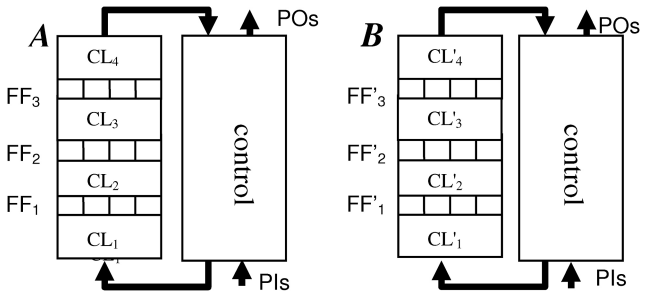

Fig. 21. B is obtained by retiming pipeline flops of A.

in the previous cycle) and $E_1 = 1$ otherwise. Each subsequent stage of the pipeline is clock-gated by delaying E_1 by one cycle using one FF. The justification for gating the pipe is given by the following theorem. We say that a stage is disabled if the enable signal to that stage is 0, i.e., the i^{th} stage FF's (FF_i) are held at their present value by disabling the clock; otherwise it is enabled.

It is helpful to recall that A^1 includes all the combinational logic in the entire pipe and not just the first stage logic, and the FFs are (FF_1, FF_2,..., FF_n) plus FFs of the FSM.

Theorem 7: Given a pipeline with depth n, with all the PIs only entering the first stage and all the POs only exiting the last stage, suppose the FSM satisfies the following properties.

1) The initial state (given to J FFs of theorem 3) of the FSM is such that all pipe stages are enabled at $t = 0$.
2) If pipe stage 1 is disabled at cycle t, the PIs must not have changed between cycles $t-1$ and t.
3) If pipe stage $j >, 1$ is disabled at cycle t, pipe stage $j-1$ must have been disabled at cycle $t-1$.

Then $\left[B = A\right]_{S_J}$.

Proof: In the following, we use $CS_j^A(t)$ to denote the state of the FF's of stage j of machine A at time t and $NS_j^A(t)$ to denote the corresponding next state of stage j.

Initially, $\left[B^1 = A^1\right]_{S_J}$ because all stages of the pipe are enabled at $t = 0$ as stated in Condition 1. Suppose stage i of machine B is disabled at time t. We will prove that stage i of machine A can be disabled at time t also. Condition 3 implies that stage $i-1$ of B must have been disabled at time $t-1$. Applying Condition 3 iteratively implies that stage 1 of machine B must have been disabled at t–(i–1). Condition 2 implies that the PIs must not have changed between t–i and t–(i–1). Since the PI's are equal at t–i and t–(i–1), the first stage of A satisfies

$$CS_1^A(t-(i-1)) = NS_1^A(t-(i-1)).$$

One cycle later this condition holds for the second stage and eventually the i^{th} stage of A satisfies $CS_i^A(t) = NS_i^A(t)$. Therefore, the i^{th} stage of A can be disabled at time t. Since a disabling at any stage of B implies a similar disabling can be done on A starting from the initial state, the two machines are equal because A can be transformed into B by this argument. ■

Note that even though the internal states of A and B always agree, the NS functions of the two machines are not equal, because $NS^A(t)$ depends on $PI(t)$ and $CS^A(t)$, while $NS^B(t)$ depends on $PI(t)$, $CS^B(t)$ and $CS^B(t-1)$.

Since the proof of the theorem uses the no-OP information coming from the fanin structure, theorem 3 should be applicable for sequential equivalence checking of this situation, i.e., to independently prove $[B = A]_{S_J}$.

To use theorem 3, the conditions to be checked are $[B^m = A^m]_{S_J}$ for some $m \geq k$ and $[A^k B^1 = A^{k+1}]$. If we set $k = m =$ number of pipe stages, then the equality $[B^k = A^k]_{S_J}$ should hold because the next state functions and outputs are unchanged when all FSM flops are initialized to enable all the stages. The equality $[A^n B^1 = A^n A^1]$ holds because in the unrolling $A^k B^1$, the outputs and final next state functions of $A^k B^1$ are independent of the flops in the FSM controller.

C. Retimed Pipelines

Another suggested application of theorem 3 is in proving equivalence of retimed pipelines in a general sequential circuit. Fig. 21 illustrates this, where a multistage pipeline is attached to a sequential circuit. B is obtained from A by retiming the pipeline FFs. A and B differ in: 1) the combinational logic between stages of the pipeline; 2) in the number of flops at each stage; and in 3) the initial states of the pipeline FFs because combinational synthesis and retiming may have been performed, and possibly iterated, when synthesizing B.

In order to prove $A = B$, we copy pipeline stages of A into B to form B' and also copy B into A to form A', as shown in Fig. 22. The output of the final stage of the dummy pipeline in A' as well as in B' is left disconnected. After the addition of dummy pipelines, the number of flip-flops in A' equals that in B'. We can now match the FF in the two circuits and apply theorem 3 where n is the depth of the pipeline. We check that B' is properly initialized and equals to A' for the first n cycles. If the other conditions of theorem 3 hold, then $A = B$.

D. Asymmetry in Verification

With respect to checking equivalence using theorem 3, we note that the addition and removal of loops are not symmetrical operations. This is illustrated by the circuit A^1 (with no loops) in Fig. 16 and B^1 (with a loop) in Fig. 17. In Fig. 19, we showed that $A^1A^1 = A^1B^1$ which is what is needed to put the loop in A. Fig. 23 shows the construction B^1B^1 and B^1A^1. We see that function $F_2(n+2)$ in B^1B^1 is dependent on $F_3(n)$ but the same function in B^1A^1 is independent of this· Thus, B^1B^1 is not equal B^1A^1 which is needed to remove the loop.

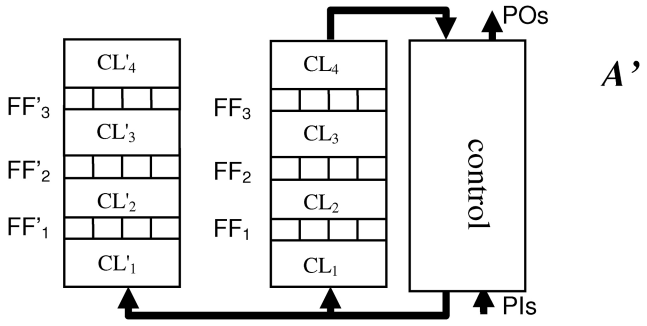

Fig. 22. Dummy pipeline added to A.

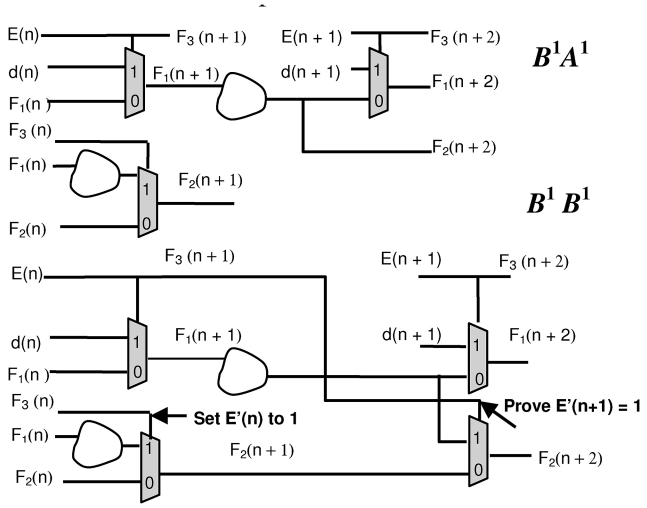

Fig. 23. Checking that $B^1B^1 = B^1A^1$.

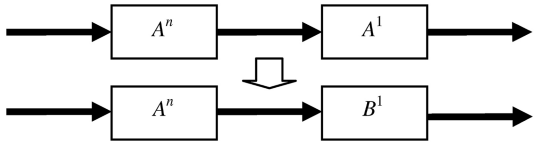

Fig. 24. Redundancy removal using SDCs from A^n to simplify A^1 resulting in B^1.

E. Redundancy Removal

A different application of theorem 3 can remove sequential redundancies. As in the ODC case, we distinguish two techniques for SDC optimization. Given A^{n+1}, the satisfiability don't cares from the first n copies of A^1 can be propagated to optimize the last copy of A^1 as shown in Fig. 24. The initial state conditions must also be correct. Given Corollary 1, $A = B$ for the following set of reachable states $R_n^A + S_J + R_1^{A,S_J} + \ldots + R_n^{A,S_J}$. Since $R_n^{A,S_J} \subset R_n^A$, R_n^{A,S_J} can be dropped. The complement of this set are the don't care conditions for optimizing A^1 into B^1. The second technique is described in the following theorem.

Theorem 8: Let B^1 be obtained from A^1 by setting a group of signals P in A^1 to constants (0 or 1). Consider B^nA^1 (first row of Fig. 25). If the initial condition $[B^n = A^n]_{S_J}$ holds, and the SDCs from B^n can be used to optimize the signals P in A^1 to be the same constants as set in B^1, then $[A = B]_{S_J}$.

Proof: By construction B^1 is optimized such that $[B^nA^1 = B^nB^1]$, and $[B^n = A^n]_{S_J}$ holds under initialization S_J.

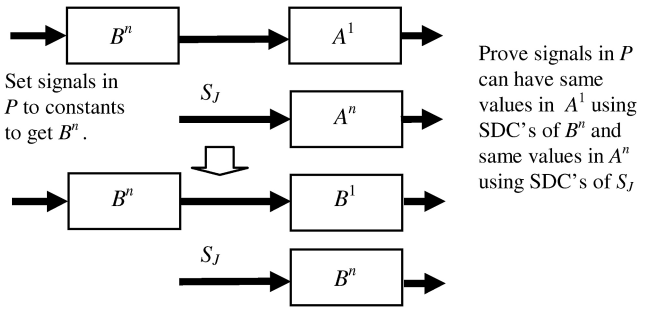

Fig. 25. Redundancy removal using SDCs.

Therefore, using theorem 3, we obtain $[A = B]_{S_J}$. Clearly, $S_J \subset R_n^B + S_J$. ■

Notice that the theorem is an instance of n-step induction used in sequential synthesis [12]. In this case, we remove sequential redundancies by first proving a change is valid for the first n cycles under the given initialization of the FFs (base case: $[B^n = A^n]_{S_J}$) and then (inductive case: $B^nA^1 \Rightarrow B^nB^1$) assuming the change holds for n consecutive cycles, prove it holds in the next cycle.

This optimization can remove redundant clock gating loops that are formed based on SDC's. For example, Circuit B in Fig. 17 has a redundant loop. The signal E' is initialized to 1 which takes care of initial conditions. If we assume $E'(n) = 1$ in the first copy of B^1 in unrolling B^1B^1 (shown in Fig. 23), we can prove $E'(n+1) = 1$ in the second copy using SDC's (When $E(n) = 0$, the inputs to MUX feeding to $F_2(n+2)$ are equal under assumption that $E'(n) = 1$). Note that the roles of A and B are swapped in applying theorem 8.

V. Using Both SDC and ODC Conditions

Theorem 1 is essentially an observability theorem, and theorems 2 and 3 are controllability theorems. One might conjecture that analogous combined controllability and observability theorems might hold. This section explores the combined approach by presenting theoretical results and providing counter examples.

Lemma 3: If $[B^1A^k = A^1A^k]_{R_n^B}$, *then* $[B^pA^k = A^pA^k]_{R_n^B}$ for any $p > 0$.

Proof: We prove this by induction on p. Clearly, this holds for $p = 1$. Assume it holds for p, i.e., $[B^pA^k = A^pA^k]_{R_n^B}$. Add A^1 to each right-hand side. Equality still holds since the states and inputs passed to A^1 are equal on both sides:

$[B^pA^kA^1 = A^pA^kA^1]_{R_n^B}$. Regroup A^1's, $[B^pA^1A^k = A^{p+1}A^k]_{R_n^B}$, and replace $[B^pA^1A^k]_{R_n^B}$ with $[B^pB^1A^k]_{R_n^B}$. This can be done because starting from a state in R_n^B any state reached after $p > 0$ copies of B^1 is still in R_n^B and $[B^1A^k = A^1A^k]_{R_n^B}$. Therefore, $[B^pB^1A^k = A^{p+1}A^k]_{R_n^B}$. Regrouping the B^1's leads to $[B^{p+1}A^k = A^{p+1}A^k]_{R_n^B}$. ■

Theorem 9: $[B^nB^1A^{k-1} = B^nA^1A^{k-1}] \Rightarrow [A = B]_{R_n^B}$.

Proof: Recall that R_n^B are the states that can be reached starting from any initial state with an n-step sequence applied to B^n. The equation $[B^nB^1A^{k-1} = B^nA^1A^{k-1}]$ can also be written as $[B^1A^{k-1} = A^1A^{k-1}]_{R_n^B}$. Suppose the theorem is false meaning $[B^1A^{k-1} = A^1A^{k-1}]_{R_n^B}$ but $[A \neq B]_{R_n^B}$. Then there

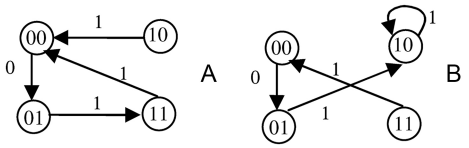

Fig. 26. Although $A^1A^1A^1 = A^1B^1A^1$, $A \neq B$ even when initialized from states in R_1^A.

exists a sequence of PIs, $\hat{I} = \{\hat{I}_1, \hat{I}_2, \ldots, \hat{I}_m, \ldots\}$, a starting state $s_0 \in R_n^B$, and an integer m, such that at clock cycle m, one of the POs of A differs from the corresponding PO of B. We know from Lemma 3 that $[B^pA^{k-1} = A^pA^{k-1}]_{R_n^B}$ for any $p > 0$ if $[B^1A^{k-1} = A^1A^{k-1}]_{R_n^B}$. This means outputs of A and B are equal for any $p>1$. Choose p large enough such that $np > m$. This contradicts the existence of sequence $\hat{I} = \{\hat{I}_1, \hat{I}_2, \ldots, \hat{I}_m, \ldots\}$ that causes a difference in the outputs. Therefore, we have a contradiction and hence $[A = B]_{R_n^B}$. ■

Note that the machine in Fig. 5, is not a counterexample to theorem 9 because $B^1B^1A^1 \neq B^1A^1A^1$, which can be seen by starting at State (01); the first produces 111 and the second produces 110.

Theorem 9 could be called a B-controllable, A-observable result while theorems 1 and 2 are A-observable and A-controllable theorems respectively. What about a formulation using both A-controllable and A-observable, say: $A^nB^1A^{k-1} = A^nA^1A^{k-1}$?

This result does not hold. To see this, consider the STG shown in Fig. 26, which has no inputs; the label on each edge denotes the output value. Although $A^1A^1A^1 = A^1B^1A^1$, one can check that$A \neq B$, even on the states that A can reach after one cycle, e.g. starting at State 01, A produces outputs 110 ... and B produces outputs 111.

Although such a theorem does not hold, the following experiment still might be useful: synthesize $A^1A^1A^1$ into $A^1B^1A^1$ to derive a new sequential machine B. It is possible that, in the synthesis into $A^1B^1A^1$, the SDCs or ODCs actually used would be produced also by B^1. So one could try to check a) $A^3 = B^1A^1A^1$ or b) $A^3 = A^1B^1B^1$ using theorems 1 or 2, or c) $B^1A^1A^1 = B^1B^1A^1$ or *d*) $A^1A^1B^1 = A^1B^1B^1$ using theorem 9[3]. If any of these cases hold, then $A = B$. For the last three, it also needs to be checked that the initial state is in the appropriate subspace or that a subset of states initialized appropriately guarantees this (theorem 3).

In summary, of the four possibilities:

1) $B^nA^k = A^{n+k} \Rightarrow A = B$;
2) $A^nB^k = A^{n+k} \Rightarrow [A = B]_{R_n^A}$;
3) $[B^nB^1A^{k-1} = B^nA^1A^{k-1}] \Rightarrow [A = B]_{R_n^B}$; and
4) $A^nB^1A^{k-1} = A^nA^1A^{k-1} \Rightarrow$,

the first three combinational results lead to a SEC conclusion, but the last one has no conclusion as far as we know. We discussed why Results 2 and 3 are only valid on a subset of initial states, which can be achieved sometimes by properly initializing a subset of flops.

[3]Note we would synthesize into $A^1B^1A^1$ because synthesizing say $A^1A^1A^1$ directly into $A^1B^1B^1$ would be difficult.

A new circuit can be derived from an original one by using both observability and controllability conditions. In such a case, theorems 1 and 2 are not necessarily applicable.

Another approach is to divide the synthesis into two steps and keep an intermediate result. Suppose A is the original circuit and B is one that has been synthesized from A by first using controllability arguments to derive intermediate circuit C. Then, B is derived from C by observability arguments. Using theorem 2, we first check that $A^nC^k = A^{n+k}$ (i.e., check the synthesis of A into C) and then by theorem 1 we check if $B^mC^j = C^{m+j}$ (i.e., check the synthesis of B into C) for some integers *n, m, j, and k*. If these checks succeed, we conclude that $[A = B]_{R_n^A}$.

VI. Relation to Previous Work

One of the pragmatic aspects of sequential synthesis is that it is insufficient to provide sequential synthesis CAD software, even if it uses formally-proved[4] transforms, because the software that embodies these may have bugs. Even if the software has been applied to many examples, most companies would insist on formally verifying the result against the original design using a third party tool. CEC, on the other hand, is practical for most industrial designs and, partly because of this, combinational synthesis is readily used in most industrial flows. Also, resolution proofs [3] can be used to produce a certificate of correctness to give further credibility to CEC.

However, the PSPACE-complete complexity of SEC often discourages the use of sequential synthesis. In special cases, the complexity of SEC can be simpler than the general case, e.g., if synthesis is restricted to one set of combinational transformations followed by one retiming (a sequential synthesis step) or vice versa, the problem is provably simpler—only NP-complete. If retiming and resynthesis are iterated, the problem is again PSPACE-complete [7] in general. Like CEC, SEC becomes simpler in practice if there are structural or functional similarities (cut-points) between the two circuits being compared.

In some cases, SEC can be transformed into a CEC problem on which today's commercial CEC engines usually can be successful, even on very large problems. One is where sequential signal equivalences (signals that are only guaranteed equivalent on the set of reachable states) are derived. For example, if induction is used in sequential synthesis and SEC is done immediately after this without other transformations intervening, SEC can be proved by iterative application of CEC methods [12].

Another example where the problem is simpler, is where a history of synthesis is recorded as a single (highly redundant) sequential circuit [9]. In most cases, this history provides a set of intermediate equivalences provable inductively, and these are usually sufficient for proving SEC. Also, the concept of speculative reduction [10] can be used to make SEC even easier in this case.

Several papers have used an (explicitly or implicitly) unrolled version of the sequential circuit to derive redundancies

[4]There are cases where proved methods in the literature have been shown to have counterexamples.

TABLE I
COMPARING THE RUNTIME OF THE PROPOSED CEC-BASED SEC AGAINST THE GENERAL SEC (THE RUNTIMES ARE IN MINUTES)

Design	Statistics				Seq-1		Seq-2	Seq-3	Seq-4	Seq-5
	Ands	FF	PI	PO	New	General	New	New	New	New
Cpu	39282	6506	51	83	0.68	15.16	0.98	1.34	1.55	1.78
ethernet	18932	10544	96	115	0.51	18.88	0.7	0.88	1.06	1.25
receiver	31103	7276	105	79	0.78	*60.55	1.29	1.51	1.69	1.63
dram_controller	81782	13822	394	703	1.61	*152.21	2.34	17.22	72.93	267.83
dm4ch	45241	11595	1741	301	0.94	25.63	1.26	1.63	2.37	**6.18
330HiFi	114824	15284	857	804	2.05	*112.83	3.26	4.09	4.79	6.22

1) * General sequence equivalence in ABC timed out. Although time-out was set to 1 hour, we were curious to see if the problem could complete if more time was given. Hence, the irregular time-out times are shown in the table.
2) ** Unresolved by ABC combinational equivalence checking.
3) Seq-*j* denotes the CEC problem where j copies of A are used, i.e. (B^1, A^j) is compared to A^{j+1}.

for synthesizing an improved circuit. However, these papers do not address the formal SEC problem of the synthesized result. All deal with the case where the redundancies derived are independent of any initial state, similar to the theorems in this paper. These types of results come mostly from the testing community, where a signal is redundant if the good and faulty (with a stuck-at fault inserted) machines cannot be distinguished no matter what initial state is used.

There is a subtle distinction between untestable faults and redundant faults. For notation, let $Z(I, s)$ be the trace of PO values generated by starting at state s, where I is the applied sequence of PI inputs. Let s_f and s_g be the initial states of the faulty and good machines respectively. Then a fault is defined to be untestable if $\forall(I)\exists(s_g, s_f)[Z^g(I, s_g) = Z^f(I, s_f)]$ and it is redundant if $\forall(I, s_f)\exists(s_g)[Z^g(I, s) = Z^f(I, s_f)]$. Using redundancy in synthesis means that when the good machine is replaced with the faulty (redundancy removed) machine, no difference can be observed externally because no matter what state, s_f, the faulty machine starts in, there is an equivalent state in which the good machine could have started in. Such a replacement is safe [5] [15] and compositional. In contrast, if the fault is merely untestable, then there could exist a pair of states in which the two machines could start, such that the difference between the two machines could not be observed. However, there could be a state in either machine which has no equivalent in the other, and if one of the machines happened to start in such a state (at power up), the two machines would have different observable behaviors. Such a (untestable) replacement is not safe and is not compositional, and its use in synthesis is problematic. A good discussion on the difference between undetectable faults and redundant faults can be found in [6].

From theorem 1, if $(B^n, A^k) = A^{n+k}$, then the synthesized circuit is a safe replacement for the original one. Safe replacements are useful because safety implies that every synchronization sequence for the original design also synchronizes the replacement. This is often desirable because, in this case, it is not necessary to rederive a new synchronizing sequence for initializing the synthesized machine.

A useful notion is c-cycle redundancy [5], where the two circuits' outputs need not match for the first c cycles after power-up. This allows more flexibility in synthesizing a circuit because the behavior of the machine need only be preserved on states that can be reached after c cycles, as long as initialization is preserved. Several papers make use of this and determine a bound k and a new circuit with the redundancies removed (called a k-delayed replacement) [8]. In [5] such redundancies are identified, one is then removed, and new ones identified. This is repeated until no more can be found. In [8], a set of "compatible" redundancies is found and removed simultaneously.

The method of [8] derives a constant n which is the difference between the time frame of an identified redundancy and the least time frame needed to infer this redundancy. Their theorem states that if the redundancy is used to create the new circuit, then it is an n-delayed replacement of the original. Note there is a difference in this result and that of theorem 2. In n-delayed replacement, it is B that is delayed for n cycles before equivalence can be proved, but in theorem 2 it is A that is delayed n cycles.

A sequential ATPG engine can be used to determine if a test vector sequence can be found which justifies a state that activates the fault in n cycles and then propagates the fault effect to a PO in k cycles. If none can be found, the fault might be redundant, but three things can go wrong with this approach: 1) undetectable faults are not necessarily redundant; 2) the justification and propagation are usually done on the good machine; and 3) finite values for n and k were used. However, such a fault is a good candidate for redundancy removal, but the result must be sequentially verified, possibly by applying theorems 1–4, which may work if A or B are projecting enough SDCs or ODCs.

An interesting discussion of incorrect proofs in the literature related to ATPG for redundancy removal can be found in [6], as well as limitations of some other methods.

VII. EXPERIMENTAL RESULTS

Some experimental results are shown in Tables I and II. Table I compares the efficiency of applying the new SEC approach of this paper against a general SEC method implemented in ABC [2]. Table II shows experiments on larger problems and compares equivalence checking after combinational clock gating against 1) sequential ODC clock gating or 2) sequential SDC clock gating.

In Table I, six large industrial benchmarks were synthesized using sequential clock-gating transforms, based on intuitively correct sequential ODC arguments. The synthesized versions

[5]A safe replacement is one for which there is no possibility of externally detecting any difference from the original.

TABLE II
COMPARING COMBINATIONAL, ODC, AND SDC VERIFICATION

Design	Cells	FF	Comb. Gating #CGs / #gated FF	Time Min.	Plus ODC gating #CGs / #gated FF	Time Min.	Plus SDC gating #CGs / #gated FF	Time Min.
Glue Logic	**5569**	**2259**	**109/1630**	**.05**	**1/156**	**.25**	**1/48**	.2
CPU Core 1	**65929**	**12554**	**341/12066**	**.5**	**20/289**	**2.5**	**11/196**	3
CPU Core 2	**126272**	**22235**	**789/22227**	**1**	**23/241**	**3.2**	**21/413**	3.5
CPU Core 3	**170287**	**39829**	**1592/39765**	**1.5**	**2/19**	**5**	**60/2707**	5.5
Video Decoder	**418275**	**62037**	**1618/61792**	**4.8**	**38/919**	**11.5**	**338/15738**	14
CPU Core 4	298500	75589	2832/75273	3	108/1988	9.2	NA	NA

1) All runtimes are on an Intel(R) Xeon(R) CPU X5570 @ 2.93GHz
2) NA means no new clock gating of the type is found.

are denoted by B and originals by A. Columns 1–5 list the sizes of the circuits. Entries in columns 6–11 are the times in minutes needed to verify equivalence. The columns *New* contain the runtime of ABC command *absec,* which proves SEC using theorem 1 and the CEC engine [11]. The column *General* denotes the use of SEC engine *dsec* [12] in ABC. Columns *seq-j* denote experiments where (B^1, A^j) was compared combinationally against A^{j+1} to show how runtimes scale as j increases. Sequential clock gating is done using(B^1, A^1)even though verification is done by comparing $(B^1, A^j)$and A^{j+1}. This is because we did not see significant increase in clock gating opportunities by adding copies of A^1during clock gating. The items marked with * or **, indicate a timeout.

Our observations from Table I are as follows.

1) In general, *New* is significantly faster than *General,* as expected (about 30 times faster for those cases when *General* could complete). The fact that *General* could actually complete on three out of the six large problems was surprising to us.
2) Except for Design 4, the new method scales approximately linearly with the size of the problem.

Table II compares the runtimes of the new SEC approach of this paper against the runtime of CEC in ABC when only combinational clocking was done. This set of experiments was done to compare the run-times of: 1) sequential clock gating and the use of theorem 1 for SEC, and 2) combinational clock gating and the use of CEC, even though the later is an easier problem. When a combinational clock gating is applied to a design, it first removes all existing clock gating elements and then extracts new ones.

Columns 2 and 3 list the size of each design. Column 4 shows the number of clock gating elements and the total number of FFs that were gated after combinational clock gating was applied. The new design is verified against the original design using CEC in ABC and the runtimes are shown in Column 5. Column 6 adds sequential ODC clock gating elements to the designs in addition to the combinational clock gating. The number of additional clock gaters and FFs affected are shown in this column. The clock gating uses ODC information from only one frame of unrolling of the design. As a result, ODC verification is done using $(B, A^1) = A^2$. The runtimes for ODC verification are shown in Column 7. Column 8 applies SDC clock gating to the designs in addition to the combinational clock gating. The SDC clock gating uses information from one previous frame. The runtimes are shown in Column 9. Our observation from Table II are as follows:

1) Runtimes of the verification approaches explained in this paper, when verification is done using one extra time frame, are only 3 to 5 times that of CEC for combinational clock gating on the same design.
2) Runtimes in Columns 7 (ODC) and 9 (SDC) are similar.

Also, although not presented in the table, we noted that the runtimes increase about 3–5x for every added frame.

VIII. CONCLUSION AND FUTURE WORK

This paper introduced several methods for sequential equivalence checking, effective in certain special cases, resulting in considerable reduction of computational effort. The methods are conservative; if the checks fail, nothing is implied about non-equivalence. Some conditions when the checks are expected to succeed include sequential clock-gating and methods that alter pipeline behavior.

Experimental results were given on twelve industrial designs that had been clock gated by an industrial tool. The SEC compared the sequentially synthesized design against the original design. It was demonstrated on six of the problems that the new SEC method was about 30 times faster than the general-case method. In addition, it was able to check three examples where general SEC could not complete. On another six problems, we compared results for purely combinational gated circuits versus when the same circuits were additionally sequentially clock gated. Run times increased only about 3x–5x for the sequential cases.

Our theorems rely on a one-to-one correspondence between the FFs of A and B. This is needed for the combinational circuits A^1B^1 or B^1A^1 to be formed where signals in the first circuit are wired to their corresponding signals in the second circuit. Some clock-gating transforms require that an enabling signal be delayed by one or more time frames, thus adding extra flops. Also in retiming, the number of flops changes. We saw in Section IV that this can be addressed by introducing dummy FFs in A, B, or both.

We conjecture, more generally, that it is sufficient to find two cuts of the same size, one in A and another in B. The signals in these cuts can be a mixture of internal wires and FF's. The only requirement is that the cuts are feedback arc sets, i.e., cutting them makes each circuit acyclic. This would allow additional applications of the theory developed in this paper to retimed circuits.

Also, it would be desirable to have a practical method to check general k-delayed equivalence, such as for designs produced by the methods of [5] and [8]. These situations are special cases where local sequential synthesis is applied. Note that if $R_k^B \subseteq R_k^A$, i.e., the set of states that are reachable in B after k steps are a subset of those of A, then theorem 2 applies and can be used to prove k-delayed equivalence. Also, it is possible that theorem 3 can be used in such cases, although experimental evaluation has not been done.

Theorem 1 legitimizes sequential synthesis based on unrolling of a sequential machine A, k times, and combinationally synthesizing the first copy of A^1 to obtain B^1 and hence a new equivalent sequential machine B. However, it is not clear what would be the result in terms of improved quality of the synthesis result.

Many industrial designs have multiple levels of clock gating. Each level adds loops to the design and potentially can block application of the theorems given in this paper to the lower levels of clock gating. The difficult cases of hierarchical clock gating are left for future research.

ACKNOWLEDGMENT

The authors would like to thank D. Berthelot of Coder Charts, and M. Bezman of Magma (currently, Synopsys) for conducting updated comparisons on industrial examples. The authors would also like to thank the industrial sponsors of BVSRC: Altera, Atrenta, Cadence, IBM, Intel, Jasper, Microsemi, Real Intent, Synopsys, Tabula, and Verific for their continued support.

REFERENCES

[1] J. Baumgartner, H. Mony, V. Paruthi, R. Kanzelman, and G. Janssen, "Scalable sequential equivalence checking across arbitrary design transformations," in *Proc. ICCD*, pp. 259–266, 2006.

[2] Berkeley Logic Synthesis and Verification Group, *ABC: A System for Sequential Synthesis and Verification*.

[3] S. Chatterjee, A. Mishchenko, R. Brayton, and A. Kuehlmann, "On resolution proofs for combinational equivalence," in *Proc. DAC*, pp. 600–605, 2007.

[4] A. P. Hurst, "Automatic synthesis of clock gating logic with controlled netlist perturbation," in *Proc. DAC*, 2008, pp. 654–657.

[5] M. A. Iyer, D. E. Long, and M. Abramovici, "Identifying sequential redundancies without search," in *Proc. DAC*, 1996, pp. 457–462.

[6] M. A. Iyer, D. E. Long, and M. Abramovici, "Surprises in sequential redundancy identification," in *Proc. EDTC*, pp. 88–94, 1996.

[7] J.-H. R. Jiang and W.-L. Hung, "Inductive equivalence checking under retiming and resynthesis," in *Proc. ICCAD*, 2007, pp. 326–333.

[8] A. Mehrotra, S. Qadeer, V. Singhal, R. K. Brayton, A. Aziz, and A. L. Sangiovanni-Vincentelli, "Sequential optimization without state space exploration," in *Proc. ICCAD*, 1997, pp. 208–215.

[9] A. Mishchenko and R. K. Brayton, "Recording synthesis history for sequential verification," in *Proc. FMCAD*, 2008, pp. 27–34.

[10] H. Mony, J. Baumgartner, V. Paruthi, and R. Kanzelman, "Exploiting suspected redundancy without proving it," in *Proc. DAC*, pp. 463–466, 2005.

[11] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een, "Improvements to combinational equivalence checking," in *Proc. ICCAD*, 2006, pp. 836–843.

[12] A. Mishchenko, M. L. Case, R. K. Brayton, and S. Jang, "Scalable and scalably-verifiable sequential synthesis," in *Proc. ICCAD*, 2008, pp. 234–241.

[13] A. Saldanha, A. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Multi-level logic optimization using don't cares and filters," in *Proc. DAC*, pp. 277–282, 1989.

[14] H. Savoj, D. Berthelot, A. Mishchenko, and R. Brayton, "Combinational techniques for sequential equivalence checking," in *Proc. FMCAD*, pp. 145–149, 2010.

[15] V. Singhal and C. Pixley, "The verification problem for safe replaceability," in *Proc. CAV*, vol. 818, 1994, pp. 311–313.

Hamid Savoj received the B.S. degree in electronics engineering from California Institute of Technology, Pasadena, CA, USA, in 1987, and the Ph.D. degree in electronics engineering and computer science from University of California, Berkeley, CA, USA, in 1992.

He co-founded Magma Design Automation, San Jose, CA, USA, an electronic design automation company, in 1997, and was the Senior Vice President of Engineering at Magma until 2006. From 2006 to 2010, he was the CTO of Envis Corporation, CA, USA, where he was focused on low power IP and software. He also co-founded Reflektion, San Mateo, CA, USA, an e-commerce company, in 2012. He is currently a visiting scholar at the University of California. He has published around 30 technical papers, and has been awarded nine patents. His current research interests include logic synthesis, formal verification, and machine learning.

Dr. Savoj is a recipient of the Artur Major Prize in Engineering from California Institute of Technology in 1987, and the Earle C. Anthony Fellowship from the University of California for the academic year 1987–1988.

Alan Mishchenko received the M.S. degree from the Moscow Institute of Physics and Technology, Moscow, Russia, in 1993, and the Ph.D. degree from the Glushkov Institute of Cybernetics, Kiev, Ukraine, in 1997.

From 1998 to 2002, he was an Intel-sponsored Visiting Scientist at Portland State University, Portland, OR, USA. Since 2002, he has been an Associate Research Engineer in the EECS Department at University of California, Berkeley, CA, USA. His current research interests include developing computationally efficient methods for synthesis and verification.

Dr. Mishchenko was a recipient of the D.O. Pederson TCAD Best Paper Award in 2008, and the SRC Technical Excellence Award in 2011, for his work on ABC.

Robert Brayton (F'70) received the Ph.D. degree in mathematics from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 1961.

He was a member of the Mathematical Sciences Department of the IBM T. J. Watson Research Center, New York, NY, USA. In 1987, he joined the Electronics Engineering and Computer Science Department at the University of California, Berkeley, CA, USA, where he is currently a Professor.

Dr. Brayton is a recipient of the IEEE Emanuel R. Piore Award in 2006, the ACM Kanallakis Award in 2006, the European DAA Lifetime Achievement Award in 2006, the EDAC/CEDA Phil Kaufman Award in 2007, the D.O. Pederson Best Paper Award in 2008, the ACM/IEEE A. Richard Newton Technical Impact in EDA Award in 2009, the Iowa State University Distinguished Alumnus Award in 2010, the SRC Technical Excellence Award in 2011, and the ACM/SIGDA Pioneering Achievement Award in 2011. He also held the Buttner Chair and the Cadence Distinguished Professorship of Electrical Engineering. He is a member of the National Academy of Engineering.